Contrôle d'accès versus Contrôle de flots

Mathieu Jaume, Valérie Viet Triem Tong, Ludovic Mé

Contrôle d'accès *versus* Contrôle de flots

Mathieu Jaume *, Valérie Viet Triem Tong,**
Ludovic Mé **

*SPI - LIP6 - UPMC,
4 Place Jussieu
75252 Paris Cedex 05
mathieu.jaume@lip6.fr,

**SUPELEC, Equipe SSIR (EA 4039),
Avenue de la Boulaie,
CS 47601 - 35576 CESSON-SEVIGNE CEDEX
prénom.nom@supelec.fr

Résumé. Traditionnellement, une politique de sécurité est mise en œuvre par un mécanisme de contrôle des accès des sujets sur les objets du système: un sujet peut lire l'information contenue dans un objet si la politique autorise ce sujet à accéder à cet objet. Une politique induit des flots d'information: si un sujet s a le droit de lire un objet o, alors toute l'information que peut un jour contenir o est accessible à s. De même, si un sujet s a le droit de modifier un objet o, alors toute l'information qui peut être portée à la connaissance de s peut se propager dans le système par le biais de o. Alors qu'une politique spécifie des autorisations sur les contenus, sa mise en œuvre contrôle les accès aux objets sans connaître leur contenu courant. Nous nous proposons dans ce travail d'étudier formellement les politiques de sécurité sous l'angle des flots d'information qu'elles induisent. Pour les politiques dont on ne peut pas montrer que tous les flots induits sont autorisés, nous définissons un mécanisme permettant de détecter les flux illégaux. Nous présentons aussi l'implémentation de ce mécanisme de détection.

1 Introduction – Motivations

La protection des informations d'un système informatique est une préoccupation majeure. L'apparition de systèmes informatiques de plus en plus grands, la dissémination de l'information et le développement des réseaux, permettent dorénavant des attaques depuis l'extérieur et rendent la protection des informations de plus en plus complexe. Nous nous intéressons ici au contrôle d'accès et au contrôle de flots d'information dans les systèmes d'information. Il s'agit de régir et de gérer les accès effectués selon certains modes (lecture, écriture, ...) par des sujets, les entités actives (processus, programmes, utilisateurs, ...) sur des objets, les entités passives (données, fichiers, programmes, ...) afin de garantir les propriétés spécifiées par une politique de sécurité. Plusieurs mécanismes peuvent être envisagés pour garantir que les accès effectués

dans un système d'information respectent la politique de sécurité souhaitée. Usuellement, on se ramène à :

- la mise en œuvre d'un moniteur de référence qui filtre les accès pour ne permettre que ceux qui ne violent pas la politique de sécurité,
- la mise en œuvre d'un système de détection d'intrusion qui repose sur l'observation des transferts d'information qui ont lieu dans le système et qui lève une alerte en cas de transfert "suspect".

Le premier moyen est une approche défensive puisqu'aucun accès non autorisé n'est possible, tandis que le second relève d'une approche préventive puisqu'elle se contente de lever un alerte en cas de doute. Généralement, ces deux mécanismes n'opèrent pas au même niveau : un moniteur de référence est habituellement défini et implanté dans un langage de haut niveau et peut être vu comme une application, tandis qu'un système de détection d'intrusion peut opérer au niveau du système d'exploitation en observant les transferts d'information engendrés lors de l'exécution d'un programme applicatif. Quoi qu'il en soit, ces deux approches ont pour vocation de garantir des propriétés de confidentialité, d'intégrité ou de confinement sur les données d'un système d'information et partagent donc un ensemble de spécifications. Dans cet article, nous étudions les liens qui existent entre ces deux mécanismes. Nous présentons donc une analyse des flots d'information associés à un modèle de contrôle d'accès. Il s'agit ici d'une interprétation de la notion de politique de contrôle d'accès différente de celle qui est implantée par un moniteur de référence. En effet, une politique de contrôle d'accès permet de spécifier quels sont les accès autorisés lorsque le système se trouve dans un certain état mais ne permet pas, tout du moins de manière explicite, de spécifier les flots d'information qui sont autorisés durant la vie du système. Ainsi, une fois qu'un sujet a pu accéder à un objet, il n'y a généralement aucun contrôle sur la propagation de l'information lue par le sujet. Par exemple, avec un modèle discrétionnaire, il est possible qu'un sujet s_1 lise un objet o_1 et recopie l'information lue dans un objet o_2 accessible en lecture par un sujet s_2 non autorisé à lire o_1. La politique de contrôle d'accès est ici respectée mais ne coïncide pas avec son interprétation en termes de flots d'information. La cohérence entre ces deux lectures sémantiques des modèles de contrôle d'accès permet d'exprimer la correspondance entre les flots engendrés par les exécutions des implantations d'un modèle de contrôle d'accès et les politiques de confidentialité, d'intégrité et de confinement induites par la politique de contrôle d'accès. Lorsque cette propriété de cohérence n'est pas vérifiée, il peut être utile de compléter le mécanisme de contrôle d'accès à l'aide d'un mécanisme de détection de flots.

Cet article propose donc une spécification des modèles de contrôle d'accès et de leurs implantations, ainsi que des flots engendrés par les exécutions de ces implantations. Ces spécifications permettent alors d'analyser l'adéquation d'une politique de contôle d'accès avec une politique de contrôle de flots et nous proposons un formalisme permettant de spécifier un mécanisme de détection de flots illégaux lorsqu'il n'y a pas adéquation entre l'interprétation en termes d'accès et en termes de flots d'une politique de contôle d'accès. Grâce à un tel cadre, nous sommes en mesure d'établir formellement les propriétés de pertinence (les flots détectés sont des flots illégaux) et de fiabilité (les flots illégaux sont détectés) d'un tel mécanisme. L'ensemble de ce travail est illustré sur un modèle discrétionnaire de contrôle d'accès. Dans Hiet et al. (2007, 2008), un prototype d'implantation d'un mécanisme de détection de flots illégaux pour ce modèle est introduit. Nous en donnons une description ainsi qu'une discussion sur son utilisation, à la fin de cet article. Le cadre formel introduit dans cet article permet, d'une part

de donner une spécification formelle de ce système, et, d'autre part, d'établir les preuves de pertinence et de fiabilité de ce système.

2 Contrôle d'accès

2.1 Modèles de contrôle d'accès

Une politique de contrôle d'accès permet de caractériser les états d'un système et de spécifier ce qu'est un état sûr en fonction d'informations de sécurité associées aux entités du système. Les entités du système peuvent être réparties dans deux ensembles : l'ensemble $\mathcal{S}$ des sujets, qui correspondent aux entités qui effectuent les actions, et l'ensemble $\mathcal{O}$ des objets, qui subissent les actions. Nous désignons par $\mathcal{A}$ l'ensemble des modes d'accès qui caractérisent les différents types d'accès effectués par les sujets sur les objets. Nous considérons ici l'ensemble $\mathcal{A} = \{\mathsf{read}, \mathsf{write}\}$. Une approche classique consiste à représenter un accès par un triplet (s, o, a), signifiant que le sujet s accède à l'objet o selon le mode d'accès a. Aussi nous notons $\mathbb{A}$ l'ensemble $\mathcal{S} \times \mathcal{O} \times \mathcal{A}$. Les systèmes de contrôle d'accès sont ici modélisés sous la forme de machines à états. Un état représente le système à un instant donné et contient au moins une description de l'ensemble des *accès courants*, c'est-à-dire de tous les accès qui ont été acceptés et qui n'ont pas encore été relâchés. Ces accès sont donc supposés être effectués simultanément dans le système. L'ensemble des états est noté Σ et l'ensemble des accès courants d'un état σ est noté $\Lambda(\sigma)$. Nous notons $\sigma \oplus (s, o, a)$ (resp. $\sigma \ominus (s, o, a)$) l'état obtenu en ajoutant (resp. en supprimant) l'accès (s, o, a) à l'ensemble des accès courants de σ. Une politique de contrôle d'accès permet de spécifier un sous-ensemble de Σ, contenant les états sûrs, c'est-à-dire les états qui vérifient la politique. Afin de déterminer si un état est sûr, les entités du système sont associées à des informations de sécurité : nous notons ρ le paramètre de sécurité qui décrit les informations de sécurité de la politique. Par exemple, le paramètre de sécurité peut être un ensemble d'accès autorisés (la matrice des droits d'accès), ou encore un treillis de niveaux de sécurité associés aux sujets et aux objets. A partir de ces informations, les états sûrs sont caractérisés par un prédicat Ω sur les états. On note $\Sigma_{|\Omega}$ l'ensemble $\{\sigma \in \Sigma \mid \Omega(\sigma)\}$ des états sûrs. Toutes ces notions permettent de définir une politique de contrôle d'accès $\mathbb{P}[\rho]$, basée sur un paramètre de sécurité ρ, comme suit : $\mathbb{P}[\rho] = (\Sigma, \Omega)$. Nous considérons dans tout cet article l'exemple de la politique HRU Harrison et al. (1976). Cette politique est paramétrée par un ensemble $m_D \subseteq \mathbb{A}$ d'accès autorisés et est notée $\mathbb{P}_{HRU}[m_D] = (\Sigma, \Omega_{HRU})$ où Ω_{HRU} caractérise les états dont les accès courants sont autorisés :

$$\Omega_{HRU}(\sigma) \Leftrightarrow \Lambda(\sigma) \subseteq m_D$$

Nous introduisons à présent la notion de modèle de contrôle d'accès, qui permet de spécifier comment passer d'un état du système à un autre. Pour cela, nous introduisons tout d'abord la notion de langage de requêtes. Une requête est soumise par un sujet afin de faire évoluer le système, soit en ajoutant, soit en enlevant un accès. Nous considérons donc ici l'ensemble de requêtes $\mathcal{R}$ contenant uniquement les deux requêtes $\langle +, s, o, a\rangle$ et $\langle -, s, o, a\rangle$, permettant de demander l'ajout (+) ou le retrait (-) d'un accès à l'ensemble des accès courants. La donnée d'une politique $\mathbb{P}[\rho]$ et d'un langage de requêtes $\mathcal{R}$ (muni d'une sémantique) constitue un modèle. Implanter un modèle consiste à définir une paire (τ, Σ_I) où $\tau : \mathcal{R} \times \Sigma \rightarrow \mathcal{D} \times \Sigma$ est une fonction de transition entre états ($\mathcal{D} = \{\mathsf{yes}, \mathsf{no}\}$ est l'ensemble des réponses possibles)

et où Σ_I est l'ensemble des états initiaux possibles (il s'agit bien sûr d'un sous-ensemble des états sûrs). Nous supposons ici que les implantations considérées sont correctes vis à vis de la politique et du langage des requêtes :

$$\begin{array}{l}\forall\sigma_1,\sigma_2\in\Sigma\ \forall R\in\mathcal{R}\ \forall d\in\mathcal{D}\quad(\Omega(\sigma_1)\wedge\tau(R,\sigma_1)=(d,\sigma_2))\Rightarrow\Omega(\sigma_2)\\ \forall\sigma_1,\sigma_2\in\Sigma\quad\tau(\langle+,s,o,a\rangle,\sigma_1)=(\mathsf{yes},\sigma_2)\Rightarrow\sigma_2=\sigma_1\oplus(s,o,a)\\ \forall\sigma_1,\sigma_2\in\Sigma\quad\tau(\langle-,s,o,a\rangle,\sigma_1)=(\mathsf{yes},\sigma_2)\Rightarrow\sigma_2=\sigma_1\ominus(s,o,a)\\ \forall\sigma_1,\sigma_2\in\Sigma\ \forall R\in\mathcal{R}\quad\tau(R,\sigma_1)=(\mathsf{no},\sigma_2)\Rightarrow\sigma_1=\sigma_2\end{array}$$

Enfin, nous notons $Exec(\tau,\Sigma_I)$ l'ensemble des séquences d'états engendrées par les exécutions de l'implantation (τ,Σ_I) :

$$Exec(\tau,\Sigma_I)=\bigcup_{n\in\mathbb{N}}\left\{\begin{array}{l}(\sigma_1,\cdots,\sigma_n)\mid\sigma_1\in\Sigma_I\wedge\\ \forall i\ (1\leq i\leq n-1)\,\exists R\in\mathcal{R}\,\tau(R,\sigma_i)=(\mathsf{yes},\sigma_{i+1})\end{array}\right\}$$

2.2 Flots et politiques de flots

Nous étudions ici les flots d'information qui se produisent entre les entités du système. Nous distinguons plusieurs sortes de flots. Nous définissons les produits cartésiens $\overset{OO}{\hookrightarrow}=\mathcal{O}\times\mathcal{O}$, $\overset{OS}{\hookrightarrow}=\mathcal{O}\times\mathcal{S}$ et $\overset{SO}{\hookrightarrow}=\mathcal{S}\times\mathcal{O}$: $o_1\overset{OO}{\hookrightarrow}o_2$ permet d'exprimer que le contenu d'un objet o_1 est propagé dans un objet o_2, $o\overset{OS}{\hookrightarrow}s$ permet d'exprimer qu'un sujet s prend connaissance des informations contenues dans un objet o et $s\overset{SO}{\hookrightarrow}o$ permet d'exprimer qu'un sujet s propage les informations dont il dispose dans un objet o. Nous caractériserons par la suite plusieurs sous-ensembles de $\overset{OO}{\hookrightarrow}$, $\overset{SO}{\hookrightarrow}$ et $\overset{OS}{\hookrightarrow}$ permettant de décrire des flots dans différents contextes. Une *politique de confinement* est un sous-ensemble $\overset{OO}{\rightsquigarrow}$ de $\overset{OO}{\hookrightarrow}$, une *politique de confidentialité* est un sous-ensemble $\overset{OS}{\rightsquigarrow}$ de $\overset{OS}{\hookrightarrow}$ et une *politique d'intégrité* est un sous-ensemble $\overset{SO}{\rightsquigarrow}$ de $\overset{SO}{\hookrightarrow}$.

Les flots d'information créés durant la vie d'un système sont caractérisés à partir d'une séquence d'états décrivant les états successifs du système. A partir de l'ensemble des accès courants qui ont lieu lorsque le système se trouve dans un état σ, nous pouvons définir les flots d'information entre objets comme suit. Le contenu d'un objet o_1 est diffusé dans un objet o_2 s'il existe un ensemble de sujets dont les accès sur les objets du système permettent de recopier l'information de o_1 dans o_2, cette information pouvant transiter par des objets intermédiaires :

$$\overset{OO}{\hookrightarrow}_\sigma=\left\{o_1\overset{OO}{\hookrightarrow}o_2\mid\left(\begin{array}{l}\exists s_1,\cdots,s_k,s_{k+1}\in\mathcal{S}\ \exists o^1,\cdots,o^k\in\mathcal{O}\\ \left\{\begin{array}{l}(s_1,o_1,\mathsf{read}),(s_1,o^1,\mathsf{write}),\\ (s_2,o^1,\mathsf{read}),(s_2,o^2,\mathsf{write}),\\ \cdots,\\ (s_i,o^{i-1},\mathsf{read}),(s_i,o^i,\mathsf{write}),\\ \cdots,\\ (s_{k+1},o^k,\mathsf{read}),(s_{k+1},o_2,\mathsf{write})\end{array}\right\}\subseteq\Lambda(\sigma)\\ \vee o_1=o_2\end{array}\right)\right\}$$

Cette définition est illustrée sur la partie gauche de la figure 1.

Nous pouvons à présent définir les flots d'information qui ont lieu durant une séquence d'états $(\sigma_1,\cdots,\sigma_n)$. Les flots entre objets sont obtenus par composition des flots engendrés

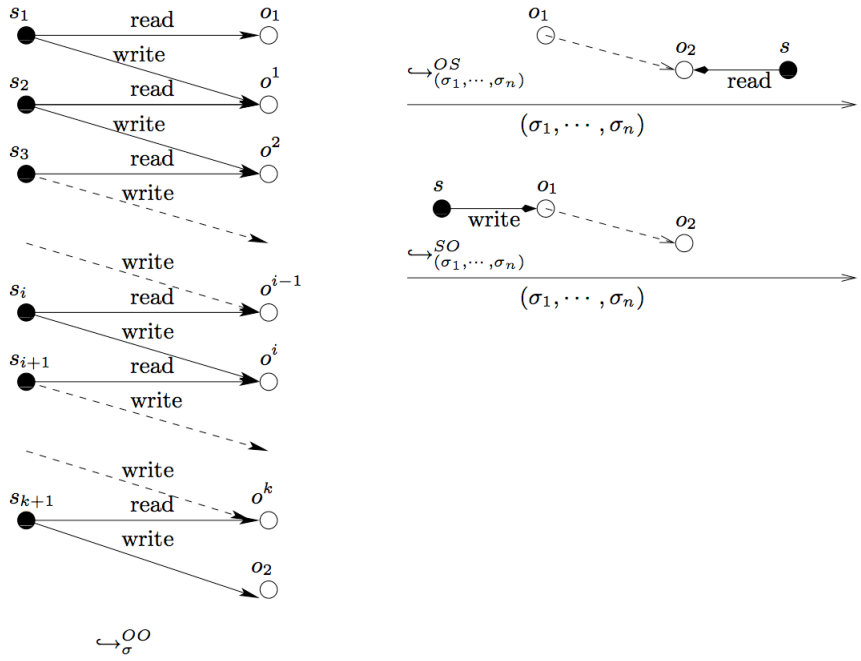

FIG. 1 – *Flots d'information*

par chacun des états apparaissant dans la séquence :

$$\overset{OO}{\hookrightarrow}_{(\sigma_1,\cdots,\sigma_n)} = \begin{cases} \overset{OO}{\hookrightarrow}_{\sigma_1} & \text{si } n = 1 \\ \overset{OO}{\hookrightarrow}_{\sigma_{k+1}} \circ \overset{OO}{\hookrightarrow}_{(\sigma_1,\cdots,\sigma_k)} & \text{si } n = k+1 \end{cases}$$

Les flots d'un objet o vers un sujet s sont identifiés lorsqu'un objet o' a reçu l'information contenue dans o et que s accède ensuite en lecture à o' :

$$\overset{OS}{\hookrightarrow}_{(\sigma_1,\cdots,\sigma_n)} = \bigcup_{i=1}^{n} \left\{ o_2 \overset{OS}{\hookrightarrow} s \mid o_2 \overset{OO}{\hookrightarrow}_{(\sigma_1,\cdots,\sigma_i)} o_1 \wedge (s, o_1, \mathsf{read}) \in \Lambda(\sigma_i) \right\}$$

Enfin, les flots d'un sujet s vers un objet o sont identifiés lorsque s accède en écriture à un objet o' dont le contenu est ensuite diffusé dans o :

$$\overset{SO}{\hookrightarrow}_{(\sigma_1,\cdots,\sigma_n)} = \bigcup_{i=1}^{n} \left\{ s \overset{SO}{\hookrightarrow} o_2 \mid (s, o_1, \mathsf{write}) \in \Lambda(\sigma_i) \wedge o_1 \overset{OO}{\hookrightarrow}_{(\sigma_i,\sigma_{i+1},\cdots,\sigma_n)} o_2 \right\}$$

Ces définitions, illustrées sur la partie droite de la figure 1, sont étendues aux ensembles de séquences d'états. Si $E \subseteq \Sigma$ est un ensemble d'états et $F \subseteq E^\star$ est un ensemble de séquences d'états, on définit :

$$\overset{X}{\hookrightarrow}_F = \bigcup_{s \in F} \overset{X}{\hookrightarrow}_s \qquad \text{où } X \in \{OO, OS, SO\}$$

Une politique de contrôle d'accès $\mathbb{P}[\rho] = (\Sigma, \Omega)$ peut être interprétée par une politique de confidentialité, une politique d'intégrité et une politique de confinement. Exprimées en termes de flots d'information, ces politiques sont définies comme suit :

$$\overset{OS}{\rightsquigarrow}_{\mathbb{P}[\rho]} = \{o \overset{OS}{\hookrightarrow} s \mid \exists \sigma \in \Sigma_{|\Omega}\ (s, o, \mathsf{read}) \in \Lambda(\sigma)\}$$
$$\overset{SO}{\rightsquigarrow}_{\mathbb{P}[\rho]} = \{s \overset{SO}{\hookrightarrow} o \mid \exists \sigma \in \Sigma_{|\Omega}\ (s, o, \mathsf{write}) \in \Lambda(\sigma)\}$$
$$\overset{OO}{\rightsquigarrow}_{\mathbb{P}[\rho]} = \left\{ \begin{array}{l} o_1 \overset{OO}{\hookrightarrow} o_2 \mid o_1 = o_2 \vee \\ \exists \sigma \in \Sigma_{|\Omega}\, \exists s \in S\, (s, o_1, \mathsf{read}) \in \Lambda(\sigma), (s, o_2, \mathsf{write}) \in \Lambda(\sigma) \end{array} \right\}$$

Ainsi, $o \overset{OS}{\rightsquigarrow}_{\mathbb{P}[\rho]} s$ signifie que pour un état sûr du système, le sujet s accède à l'information contenue dans l'objet o et $s \overset{SO}{\rightsquigarrow}_{\mathbb{P}[\rho]} o$ signifie que pour un état sûr du système, le sujet s propage l'information dont il dispose dans l'objet o. Il est à présent possible d'exprimer formellement la propriété de cohérence entre les deux interprétations d'un modèle de contrôle d'accès :

$$\overset{OS}{\hookrightarrow}_{Exec(\tau,\Sigma^I)} \subseteq \overset{OS}{\rightsquigarrow}_{\mathbb{P}[\rho]} \wedge \overset{SO}{\hookrightarrow}_{Exec(\tau,\Sigma^I)} \subseteq \overset{SO}{\rightsquigarrow}_{\mathbb{P}[\rho]} \wedge \overset{OO}{\hookrightarrow}_{Exec(\tau,\Sigma^I)} \subseteq \overset{OO}{\rightsquigarrow}_{\mathbb{P}[\rho]}$$

Cette propriété exprime donc que tous les flots engendrés par les exécutions des implantations correctes d'un modèle de contrôle d'accès, respectent l'interprétation en termes de flots de ce modèle. Il a été prouvé que cette propriété de cohérence est vérifiée pour les politiques de la Muraille de Chine Brewer et Nash (1989) et de Bell et LaPadula LaPadula et Bell (1996); Bell et LaPadula (1973) (qui peuvent donc être vues comme de véritables politiques de flots). En revanche, cette propriété n'est pas vérifiée pour les modèles HRU et RBAC (*Role-Based Access Control*) Sandhu et al. (1996). Toutefois, pour ces deux modèles, les propriétés permettant de garantir la cohérence ont été caractérisées : elles portent sur les informations de sécurité propres à ces modèles (matrice de droits d'accès, hiérarchie de rôles, permissions).

3 Contrôle de flots

3.1 Mécanismes de détection de flots

Lorsque la propriété de cohérence n'est pas satisfaite, on peut adjoindre un mécanisme de détection de flots au mécanisme de contrôle d'accès. Etant donné un ensemble de flots $\rightsquigarrow_{\mathbb{F}}$, détecter les flots appartenant à $\rightsquigarrow_{\mathbb{F}}$ lors de la vie d'un système consiste à observer les séquences d'états du système et à collecter des informations permettant d'identifier celles d'entre elles qui engendrent des flots recherchés. C'est à partir de cette information que l'on peut définir un prédicat $\mho$ sur Σ permettant de caractériser les états issus de séquences d'états engendrant un flot appartenant à $\rightsquigarrow_{\mathbb{F}}$. Ces états sont appelés des états d'alerte et on note $\Sigma_{|\mho}$ l'ensemble $\{\sigma \in \Sigma \mid \mho(\sigma)\}$. Etant donnés un ensemble E d'états observables et un ensemble $F \subseteq E^\star$ de séquences d'états qui peuvent se produire, un mécanisme de détection de flots est défini par : $\mathbb{F}[\rho, E, F, \rightsquigarrow_{\mathbb{F}}] = (\Sigma, \mho)$.

Bien sûr, il faut s'assurer que le prédicat $\mho$ caractérise bien les états issus d'une séquence produisant au moins un flot dans $\rightsquigarrow_{\mathbb{F}}$. Pour cela, nous introduisons les deux propriétés classiques suivantes. $\mathbb{F}[\rho, E, F, \rightsquigarrow_{\mathbb{F}}] = (\Sigma, \mho)$ est :

– *pertinent* ssi tout état d'alerte est issu d'une séquence engendrant un flot dans $\rightsquigarrow_{\mathbb{F}}$:

$$\forall(\sigma_1, \cdots, \sigma_n) \in F \quad \mho(\sigma_n) \Rightarrow \overset{X}{\hookrightarrow}_{(\sigma_1,\cdots,\sigma_n)} \cap \rightsquigarrow_{\mathbb{F}} \neq \emptyset$$

– *fiable* ssi tout état issu d'une séquence engendrant un flot dans $\rightsquigarrow_{\mathbb{F}}$ est un état d'alerte :

$$\forall(\sigma_1, \cdots, \sigma_n) \in F \quad \overset{X}{\hookrightarrow}_{(\sigma_1,\cdots,\sigma_n)} \cap \rightsquigarrow_{\mathbb{F}} \neq \emptyset \Rightarrow \mho(\sigma_n)$$

où $X \in \{OO, OS, SO\}$.

3.2 Définition d'un mécanisme de contrôle de flots

Nous présentons ici l'utilisation du formalisme qui vient d'être présenté pour modéliser le mécanisme de détection d'intrusions défini dans Hiet et al. (2007) ainsi que les preuves de fiabilité et de pertinence de ce mécanisme. Le mécanisme de détection de flots que nous introduisons ici permet de détecter les flots engendrés par des séquences d'états produisant des flots ne respectant pas une certaine politique de flots $\overset{OO}{\rightsquigarrow}$ entre objets. Ce mécanisme peut être aussi utilisé pour des politiques de flots entre sujets et objets. Il suffit pour cela d'associer à chaque sujet s un objet o_s. Dans ce cas, les objets du système sont les objets de l'ensemble $\mathcal{O}$ et les objets d'un ensemble $\mathcal{O}_{\mathcal{S}}$ contenant les objets associés aux sujets. Bien sûr, les objets considérés par le prédicat de sécurité sont les objets de $\mathcal{O}$ et pour tout état σ, nous supposons :

$$\{(s, o_s, \mathsf{read}), (s, o_s, \mathsf{write})\} \subseteq \Lambda(\sigma) \text{ et } ((s, o, a) \in \Lambda(\sigma) \wedge o \in \mathcal{O}_{\mathcal{S}}) \Rightarrow o = o_s \qquad (1)$$

On suppose donc que, quelque soit l'état du système, tout sujet s a toujours un accès en lecture et en écriture sur l'objet o_s qui lui est associé, et que seul s peut avoir un accès sur cet objet o_s. Dans ce cadre, on montre facilement que :

$$\overset{OS}{\hookrightarrow}_{(\sigma_1,\ldots,\sigma_n)} = \{o_1 \overset{OO}{\hookrightarrow}_{(\sigma_1,\ldots,\sigma_n)} o_2 \mid o_1 \in \mathcal{O} \wedge o_2 \in \mathcal{O}_{\mathcal{S}}\}$$
$$\overset{SO}{\hookrightarrow}_{(\sigma_1,\ldots,\sigma_n)} = \{o_1 \overset{OO}{\hookrightarrow}_{(\sigma_1,\ldots,\sigma_n)} o_2 \mid o_1 \in \mathcal{O}_{\mathcal{S}} \wedge o_2 \in \mathcal{O}\}$$

En effet, avec l'hypothèse (1), les flots entre un sujet s et un objet o peuvent être obtenus en considérant les flots entre o et l'objet o_s associé à s.

Nous proposons un mécanisme permettant de détecter les flots d'information interdits par une politique $\overset{OO}{\rightsquigarrow}$ entre objets. Ce mécanisme est noté $\mathbb{F}[\rho, E, F, \rightsquigarrow_{\mathbb{F}}]$, il est paramétré par un paramètre de sécurité ρ, par un ensemble d'états observables E, et par un ensemble de séquences d'états $F \subseteq E^\star$ tel que pour toute séquence $(\sigma_1, \cdots, \sigma_n) \in F$:

– aucun accès n'est effectué dans l'état initial σ_1 : $\Lambda(\sigma_1) = \emptyset$
– l'état σ_{i+1} est obtenu à partir de l'état σ_i en ajoutant ou en supprimant un accès de l'ensemble des accès courants de σ_i :

$$\forall(\sigma_1, \cdots, \sigma_n) \in F \; \forall i \in \mathbb{N} \; (1 \leq i \leq n-1)$$
$$\sigma_{i+1} = \sigma_i \oplus (s, o, a) \vee \sigma_{i+1} = \sigma_i \ominus (s, o, a)$$

Notre mécanisme de détection associe deux méta-données (ou tags) à chaque objet. Le premier tag d'un objet $o \in \mathcal{O}$ represente l'origine de l'information contenue dans o. Ce premier

tag est une collection d'objets de $\mathcal{O}$ tel que le contenu courant de o dérive des objets contenus dans cette collection (voir (2)). Ce premier tag est appelé tag *information* puisqu'il permet de caractériser l'origine de l'information contenue. Le tag *information* d'un objet o dans un état σ du système est noté $T^I_\sigma(o)$. Le second tag d'un objet caractérise les ensembles d'objets dont le contenu est directement autorisé à se propager dans o par la politique $\overset{OO}{\rightsquigarrow}$ (voir 3). Ce second tag dérive de l'expression de la politique $\overset{OO}{\rightsquigarrow}$, il est appelé tag *politique* d'un objet o et est noté $T^P_\sigma(o)$. Plus formellement, nous construisons les tags de façon à ce qu'ils assurent les propriétes suivantes :

$$\forall \sigma_n \, T^I_{\sigma_n}(o) \quad = \quad \{o' |\, o' \overset{OO}{\hookrightarrow}_{(\sigma_1,\cdots,\sigma_n)} o\} \tag{2}$$

$$\forall \sigma_n \, T^P_{\sigma_n}(o) \quad = \quad \{o' \in O |\, (o = o') \vee (o' \overset{OO}{\rightsquigarrow} o)\} \tag{3}$$

Ces tags sont initialisés de façon à assurer ces propriétés et à chaque changement d'état qui induit un flot d'information, nous modifions les tags de manière à préserver les propriétés.

3.3 Initialisation des tags

Pour tout objet o et pour tout état σ_1 débutant une séquence, le tag *information* est réduit à l'ensemble contenant o lui-même si $o \notin \mathcal{O}_\mathcal{S}$. Pour les objets $o \in \mathcal{O}_\mathcal{S}$ le tag information est initialement vide car nous faisons l'hypothèse que les informations du système sont uniquement celles contenues dans des objets non associés à des utilisateurs et donc n'appartenant pas à $\mathcal{O}_\mathcal{S}$. Pour tous les objets, le tag *politique* est la collection des objets dont le contenu initial est autorisé par $\overset{OO}{\rightsquigarrow}$ à se propager dans o. Plus formellement, ces tags se définissent comme suit :

$$T^I_{\sigma_1}(o) = \emptyset \text{ si } o \in \mathcal{O}_\mathcal{S} \text{ sinon } T^I_{\sigma_1}(o) = \{o\}$$
$$\text{et } T^P_{\sigma_1}(o) = \{o' |\, o' \overset{OO}{\rightsquigarrow} o\}$$

Il est clair que, par construction, les tags assurent les propriétés (2) et (3).

3.4 Evolution des tags

Les tags sont mis à jour à chaque changement d'état : lorsque le système évolue d'un état σ_i vers un état σ_{i+1}, les accès courants sur les objets sont modifiés, ce qui impacte les flots d'information (voir section 2.2). Les tags *politiques* ne sont jamais modifiés : en effet, le tag *politique* reflète les flots autorisés par la politique $\overset{OO}{\rightsquigarrow}$ et cette politique est considérée comme fixe dans ce travail. En revanche, le tag *information* d'un objet reflète les informations qui se sont propagées dans cet objet au cours de la séquence. Lors d'un changement d'état, et en particulier lors d'un changement d'état induit par un ajout d'accès (lecture ou écriture), de nouveaux flots d'information sont possibles. Le tag information de chaque objet dont le contenu est impacté par le changement est donc modifié de façon à assurer le respect de la propriété (2). Formellement, si le système évolue d'un état σ_i vers un état σ_{i+1} et si σ_{i+1} a été obtenu en ajoutant un accès aux accès courants de σ_i, alors nous modifions le tag information des objets $o \in \mathcal{O} \cup \mathcal{O}_\mathcal{S}$:

$$T^I_{\sigma_{i+1}}(o) = T^I_{\sigma_i}(o) \cup \bigcup_{\{o' |\, o' \overset{OO}{\hookrightarrow}_{\sigma_{i+1}} o\}} T^I_{\sigma_i}(o')$$

Si σ_{i+1} a été obtenu en supprimant un accès aux accès courants de σ_i, alors les "tags" ne changent pas. Remarquons que les objets dont le contenu est impacté par un changement d'état (de σ_i vers σ_{i+1}) sont bien les objets o tels que $\{o' |\, o' \overset{OO}{\hookrightarrow}_{\sigma_{i+1}} o\}$ est un ensemble non vide ; les tags informations des autres objets ne changent pas.

3.5 Détection des flots illégaux

Le système de "tags" permet de définir le prédicat $\mho$ caractérisant les états d'alerte comme suit :

$$\mho(\sigma) \Leftrightarrow \exists o \in \mathcal{O} \cup \mathcal{O}_{\mathcal{S}}\ T^I_\sigma(o) \not\subseteq T^P_\sigma(o)$$

Nous montrons alors que le mécanisme de détection de flots ainsi obtenu est pertinent et fiable.

Proposition 1 $\mathbb{F}[\rho, E, F, \rightsquigarrow_{\mathbb{F}}] = (\Sigma, \mho)$ *est pertinent et fiable.*

3.6 Application

Nous illustrons ici l'analyse des flots d'information d'un modèle de contrôle d'accès et la mise en œuvre d'un mécanisme de détection de flots en considérant le modèle HRU. L'instance de la politique que nous considérons ici repose sur l'ensemble m_D des accès autorisés, représenté par la matrice de droits d'accès suivante :

	o_1	o_2	o_3	o_4
Alice	read, write		read	
Bob	read	read, write		
Charlie		read, write		write

Politiques de confidentialité et d'intégrité. Les politiques de confidentialité et d'intégrité, exprimées en termes de flots d'information, induites pas la politique de contrôle d'accès, sont définies comme suit :

$$\overset{OS}{\rightsquigarrow}_{\mathbb{P}_{HRU}[m_D]} = \left\{ o_1 \overset{OS}{\hookrightarrow} \text{Alice}, o_3 \overset{OS}{\hookrightarrow} \text{Alice}, o_1 \overset{OS}{\hookrightarrow} \text{Bob}, o_2 \overset{OS}{\hookrightarrow} \text{Bob}, o_2 \overset{OS}{\hookrightarrow} \text{Charlie} \right\}$$

$$\overset{SO}{\rightsquigarrow}_{\mathbb{P}_{HRU}[m_D]} = \left\{ \text{Alice} \overset{SO}{\hookrightarrow} o_1, \quad \text{Bob} \overset{SO}{\hookrightarrow} o_2, \quad \text{Charlie} \overset{SO}{\hookrightarrow} o_2, \quad \text{Charlie} \overset{SO}{\hookrightarrow} o_4 \right\}$$

Les propriétés de cohérence ne sont pas vérifiées pas vérifiée par les exécutions de $(\tau_{HRU}, \Sigma^I_{HRU})$. En effet, si l'on considère par exemple l'ensemble m_D introduit précédemment, nous avons :

$$o_3 \overset{OS}{\hookrightarrow}_{Exec(\tau_{HRU}, \Sigma^I_{HRU})} \text{Bob et Bob} \overset{SO}{\hookrightarrow}_{Exec(\tau_{HRU}, \Sigma^I_{HRU})} o_4$$

car Alice peut lire o_3 et écrire dans o_1 qui est accessible en lecture par Bob et car Bob peut écrire dans o_2 et Charlie peut lire o_2 et écrire dans o_4, mais nous n'avons ni $o_3 \overset{OS}{\rightsquigarrow}_{\mathbb{P}'_{HRU}[m_D]}$ Bob, ni Bob $\overset{SO}{\rightsquigarrow}_{\mathbb{P}'_{HRU}[m_D]} o_4$.

Puisque les exécutions de $(\tau_{HRU}, \Sigma^I_{HRU})$ ne sont pas cohérentes pour les politiques de flots $\overset{OS}{\rightsquigarrow}_{\mathbb{P}_{HRU}[m_D]}$ et $\overset{SO}{\rightsquigarrow}_{\mathbb{P}_{HRU}[m_D]}$, nous utilisons maintenant le mécanisme de détection de flots afin que le système lève une alerte dès qu'il détecte un flot ne respectant pas ces politiques. Ici, ces deux politiques de flots sont exprimées en associant un objet o_s à chaque sujet s. Plus formellement, le mécanisme de détection de flots est obtenu en considérant :

- l'ensemble m_D comme paramètre de sécurité ρ,
- l'ensemble $\Sigma_{|\Omega_{HRU}}$ comme ensemble d'états observables E,
- l'ensemble $Exec(\tau_{HRU}, \Sigma^I_{HRU})$, où $\Sigma^I_{HRU} \subseteq \{\sigma \in \Sigma \mid \Lambda(\sigma) = \emptyset\}$, comme ensemble de séquences d'états observables,
- la relation :

$$\overset{OO}{\rightsquigarrow} = \left\{ \begin{array}{lll} o_1 \overset{OO}{\hookrightarrow} o_2 \mid & & (o_1 \in \mathcal{O} \wedge o_2 = o_s \in \mathcal{O}_\mathcal{S} \wedge o_1 \overset{OS}{\rightsquigarrow}_{\mathbb{P}_{HRU}[m_D]} s) \\ & \vee & (o_1 = o_s \in \mathcal{O}_\mathcal{S} \wedge o_2 \in \mathcal{O} \wedge s \overset{SO}{\rightsquigarrow}_{\mathbb{P}_{HRU}[m_D]} o_2) \end{array} \right\}$$

  comme politique de flots $\overset{OO}{\rightsquigarrow}$.

La proposition 1 nous garantit alors que les états d'alerte sont exactement les états issus de séquences engendrant un flot ne respectant pas les politiques $\overset{OS}{\rightsquigarrow}_{\mathbb{P}_{HRU}[m_D]}$ et $\overset{SO}{\rightsquigarrow}_{\mathbb{P}_{HRU}[m_D]}$.

Par exemple, considérons à nouveau le modèle HRU introduit précédemment et en particulier considérons la séquence d'états $(\sigma_1, \sigma_2, \sigma_3, \sigma_4)$ telle que

- σ_1 est un état dont l'ensemble des accès courants est vide,
- $\sigma_2 = \sigma_1 \oplus (Alice, o_3, read)$,
- $\sigma_3 = \sigma_2 \oplus (Alice, o_1, write) = \sigma_1 \oplus (Alice, o_3, read) \oplus (Alice, o_1, write)$
- $\sigma_4 = \sigma_3 \oplus (Bob, o_1, read) = \sigma_1 \oplus (Alice, o_3, read) \oplus (Alice, o_1, write) \oplus (Bob, o_1, read)$

L'ensemble des objets est $\mathcal{O} = \{o_1, o_2, o_3, o_4, o_A, o_B, o_C\}$ où o_A, o_B et o_C sont les objets respectivement associés aux sujets Alice, Bob et Charlie. D'après la construction décrite en 3.3, dans l'état initial σ_1 ces objets sont associés chacun à deux tags qui évoluent ensuite selon la construction présentée en 3.4. La figure 3.6 présente l'initialisation et l'évolution des tags de chacun de ces objets : les objets sont présentés en colonne, chaque ligne du tableau présente les valeurs des tags dans les différents états σ_i (i =1,2,3,4).

Comme attendu, les états σ_1, σ_2 et σ_3 ne sont pas des états d'alertes, le lecteur peut vérifier que pour chaque objet, et chaque état σ_1, σ_2 et σ_3, le tag information de l'objet est inclus dans le tag politique de l'objet. Au contraire dans l'état σ_4, le tag information de l'objet o_B n'est pas inclus dans son tag politique. σ_4 est un état d'alerte ce qui correspond bien au fait qu'il existe un chemin pour l'information permettant à Bob d'accèder au contenu de l'objet o_3 alors que la politique ne l'y autorise pas.

4 Implantation

Le système de tags présenté ci-dessus peut être implanté à divers niveaux de granularité. Dans Hiet et al. (2007), nous avons présenté une implantation au niveau du système d'exploitation (Blare) et dans Hiet et al. (2008) une implantation au niveau de la machine virtuelle Java (JBlare). Pour illustrer la mise en œuvre du système de tag présenté ci-dessus, nous proposons ici, à titre d'exemple, une bref résumé de l'architecture et du fonctionnement de Blare, le détecteur implanté au niveau système d'exploitation.

	o_1	o_2	o_3	o_4	o_A	o_B	o_C
$T^I_{\sigma_1}()$	$\{o_1\}$	$\{o_2\}$	$\{o_3\}$	$\{o_4\}$	$\{\}$	$\{\}$	$\{\}$
$T^P_{\sigma_1}()$	$\{o_1, o_3\}$	$\{o_1, o_2, o_4\}$	$\{o_3\}$	$\{o_2, o_4\}$	$\{o_1, o_3\}$	$\{o_1, o_2\}$	$\{o_2\}$
$\sigma_2 = \sigma_1 \oplus (\mathbf{Alice}, \mathbf{o_3}, \mathbf{read})$							
$T^I_{\sigma_2}()$	$\{o_1\}$	$\{o_2\}$	$\{o_3\}$	$\{o_4\}$	$\{o_3\}$	$\{\}$	$\{\}$
$T^P_{\sigma_2}()$	$\{o_1, o_3\}$	$\{o_1, o_2, o_4\}$	$\{o_3\}$	$\{o_2, o_4\}$	$\{o_1, o_3\}$	$\{o_1, o_2\}$	$\{o_2\}$
$\sigma_3 = \sigma_2 \oplus (\mathbf{Alice}, \mathbf{o_1}, \mathbf{write})$							
$T^I_{\sigma_3}()$	$\{o_1, o_3\}$	$\{o_2\}$	$\{o_3\}$	$\{o_4\}$	$\{o_3\}$	$\{\}$	$\{\}$
$T^P_{\sigma_3}()$	$\{o_1, o_3\}$	$\{o_1, o_2, o_4\}$	$\{o_3\}$	$\{o_2, o_4\}$	$\{o_1, o_3\}$	$\{o_1, o_2\}$	$\{o_2\}$
$\sigma_4 = \sigma_3 \oplus (\mathbf{Bob}, \mathbf{o_1}, \mathbf{read})$							
$T^I_{\sigma_4}()$	$\{o_1, o_3\}$	$\{o_2\}$	$\{o_3\}$	$\{o_4\}$	$\{o_3\}$	$\{o_1, o_3\}$	$\{\}$
$T^P_{\sigma_4}()$	$\{o_1, o_3\}$	$\{o_1, o_2, o_4\}$	$\{o_3\}$	$\{o_2, o_4\}$	$\{o_1, o_3\}$	$\{o_1, o_2\}$	$\{o_2\}$

FIG. 2 – *Evolution des tags suivant la séquence d'accès décrite en 3.6*

La figure 3 propose une vue générale de l'architecture de Blare. Le détecteur en lui-même est entièrement implanté dans l'espace protégé du système d'exploitation (espace noyau). Des outils s'exécutant dans l'espace non-protégé (espace utilisateur) ont aussi été développés, ainsi qu'une librairie associée qui constitue une interface pour la programmation d'applications (API). Le rôle des différents éléments de cette architecture est décrit ci-après. Blare est fourni sous la forme de fichiers sources modifiés du noyau Linux ou sous la forme d'un *patch* [1].

Capture des flux d'informations. Afin de détecter les accès aux objets du système d'exploitation, des points d'insertion ou *hooks* ont été insérés dans certaines fonctions du noyau, ce qui permet d'éviter le surcoût classique dû au déroutement des appels système. Ces points d'insertions sont principalement situés dans la couche *Virtual File System* (système de fichier virtuel, VFS) du noyau. Comme c'est traditionnellement le cas pour les systèmes de type UNIX, Linux fournit un accès aux objets du système via la notion de fichiers. Le *VFS* constitue donc une abstraction unifiée, sous forme de fichiers, des différents types d'objets du système. Par exemple, les fichiers provenant des différents systèmes de fichiers, certaines variables noyau du système virtuel `/proc`, les partages réseaux, les sessions TCP, etc. peuvent tous être accédés via l'interface générique du *VFS*. Certains objets, comme les connexions réseau via le mécanisme de *socket* ou les connexions sur une console, nécessitent en outre le recours à des fonctions qui sortent du cadre du *VFS*. Le code source du noyau qui implémente ces fonctions a donc également du être modifié.

Ces modifications se traduisent principalement par un appel à l'algorithme générique de détection d'intrusions via deux nouvelles fonctions du noyau qui modélisent les flux élémentaires d'information comme expliqué ci-après. Lorsqu'un processus utilisateur accède à un objet, il

1. http ://www.rennes.supelec.fr/blare/download.html

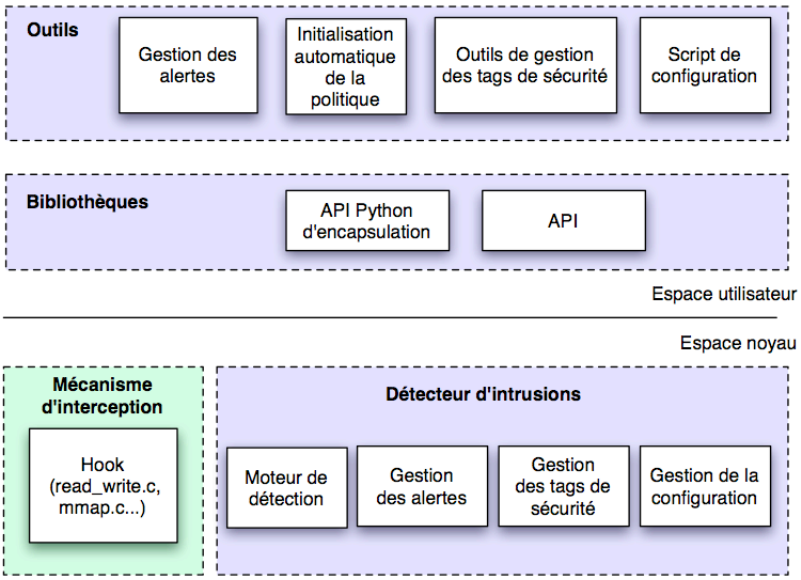

FIG. 3 – *Architecture du détecteur Blare*

doit invoquer l'appel système approprié et effectue par conséquent un appel à nos fonctions modifiées. Ainsi, tout flux d'informations généré au sein du système est vérifié.

Vérification de la légalité des flux. Les fonctionnalités du mécanisme de détection peuvent être séparées en quatre classes (voir figure 3) :

1. le cœur du mécanisme de détection, le moteur de détection, implémente le modèle de détection présenté dans cet article. Un tag politique est associé à chaque objet et un tag information à son contenu courant, sous la forme, dans les attributs étendus, de deux listes chaînées d'entiers (un entier est initialement associé au contenu initial de chaque objet). Ces listes sont utilisées pour la vérification de la légalité d'un flux : les entiers de la liste implantant le tag information doivent appartenir à la liste implantant le tag politique. Si ce n'est pas le cas, une alerte est émise. Cette vérification est faite au travers de l'invocation, par chaque appel système qui engendre un flux d'information, de fonctions de vérification ajoutées par nos soin au noyau. Ces fonctions mettent également à jour les tags en fonction des flux d'information observés. Une première fonction vérifie les flux simples qui mettent uniquement en œuvre un objet en lecture et un objet en écriture, une autre fonction vérifie les flux plus complexes où plusieurs objets sont accédés en lecture. On notera que, sous Linux, il n'existe pas de flux avec plusieurs destinations.
2. le composant de gestion de la configuration permet à des processus utilisateurs d'administrer le détecteur ou de vérifier certaines données telles que le nombre total d'alertes. Ces fonctionnalités sont réalisées via l'interface *sysfs* de Linux. Elles s'avèrent parti-

culièrement utiles pour les tâches de diagnostic des alertes ou de mise au point, mais peuvent bien sûr être verrouillées pour des raisons de sécurité ;

3. le composant de gestion des tags de sécurité permet aux utilisateurs de spécifier la politique de sécurité en associant des tags aux objets et à leur contenu initial ;
4. le composant de gestion des alertes incrémente le compteur des alertes, avertit un utilitaire s'exécutant en espace utilisateur dès qu'une alerte est émise et peut éventuellement enregistrer directement dans un fichier la succession des alertes.

Comme l'administrateur du système doit spécifier la politique de sécurité sous forme de tags pour chaque conteneur (par exemple, pour les fichiers et les *socket*), une interface permettant d'accéder aux tags depuis l'espace utilisateur a été implanté. En outre, afin de faciliter cette tâche d'administration, une interface de programmation d'applications comprenant des fonctions de plus haut niveau est fournie, ainsi que des utilitaires permettant de gérer directement les tags depuis l'espace utilisateur. L'administrateur peut donc spécifier ou consulter les tags d'un fichier ou d'une *socket* d'une façon similaire à l'utilisation de l'outil UNIX traditionnel. De plus, un *wrapper* de l'API pour le langage Python a été développé, permettant aux développeurs d'utiliser facilement les fonctions de l'API dans des scripts Python. Ce *wrapper* est notamment utilisé par un script de génération automatique de la politique à partir des droits DAC.

5 Conclusion

La sécurité, et plus particulièrement le contrôle d'accès, sont des problématiques actuelles en informatique. En effet, il devient aujourd'hui important de pouvoir contrôler les flots d'information dans les réseaux et dans les systèmes d'information. Il convient donc de développer au sein des systèmes informatiques des mécanismes permettant de filtrer les accès afin de ne laisser passer que ceux autorisés. La conception et le développement de ces mécamismes doivent être menés de manière à garantir leur fiabilité et leur sûreté. L'emploi des méthodes formelles dans le développement d'un moniteur de référence permet de garantir que certaines propriétés de sécurité sont toujours respectées. Toutefois, une politique de contrôle d'accès ne fournit généralement pas explicitement de propriétés sur la dissémination des informations dans le système et afin de garantir qu'une certaine politique de flots est respectée, il peut être utile d'adjoindre un mécanisme de détection de flots au moniteur de référence. Dans cet article, nous avons présenté un ensemble de spécifications formelles permettant d'adresser ce problème et de fournir un cadre dans lequel il est possible d'analyser formellement les flots engendrés par l'implantation d'un modèle de contrôle d'accès et de vérifier leur cohérence avec les politiques de flots induites par la politique de contrôle d'accès. Un mécanisme de détection de flots illégaux a été proposé afin de prendre en compte les situations où cette cohérence n'est pas vérifiée. Une implantation a également été décrite.

Une importante littérature sur les flots d'information existe. Dans Denning (1976a), D.E. Denning introduit une notion de modèle de flots d'information et présente dans Denning (1976b) un procédé de transformation de politiques de flots, satisfaisant certaines conditions, en une politique de flots ayant une structure de treillis (et pouvant donc être mise en œuvre par une politique de contrôle d'accès *à la* Bell et LaPadula). Dans Denning et Denning (1977), D.E.

Denning utilise ce même modèle pour définir un mécanisme de certification de programmes garantissant des propriétés sur les flots engendrés lors de l'exécution de programmes. Dans Foley (1990), S.N. Foley introduit un modèle de flots permettant de prendre en compte des politiques de flots non nécessairement transitives et illustre l'intérêt d'un tel modèle sur des exemples concrets. Les liens entre modèles de contrôle d'accès et modèles de contrôle de flots ont également été étudiés. Dans McLean (1990), J. McLean introduit une théorie des flots d'information afin de définir un modèle de sécurité à base de flots. Dans Osborn (2002), S.L. Osborn décrit comment caractériser les flots possibles dans un système RBAC à partir des paramètres de sécurité de ce système. Plus récemment, dans Ayed et al. (2007) les auteurs formalisent la notion de DTE (*Domain Type Enforcement*) Badger et al. (1995) afin de fournir un mécanisme de contrôle de flots qu'ils intègrent ensuite dans le modèle OrBAC (*Organization based access control model*) Abou El Kalam et al. (2003). Ici encore, tous ces travaux portent sur des notions communes et il serait intéressant de les exprimer dans notre formalisme afin de pouvoir analyser leurs points communs et leurs différences, de comparer leur pouvoir d'expression et de comprendre comment une implantation de l'un de ces modèles peut être étendue pour inclure les particularités d'un autre modèle.

Remerciements. Ce travail est partiellement financé par l'action ANR SSURF.

Références

Abou El Kalam, A., S. Benferhat, A. Miège, R. El Baida, F. Cuppens, C. Saurel, P. Balbiani, Y. Deswarte, et G. Trouessin (2003). Organization based access control. In *IEEE 4th International Workshop on policies for distributed systems and networks*, pp. 120. IEEE Computer Society.

Ayed, S., S. Cuppens-Boulahia, et F. Cuppens (2007). An integrated model for access control and information flow requirements. In I. Cervesato (Ed.), *ASIAN*, Volume 4846 of *Lecture Notes in Computer Science*, pp. 111–125. Springer.

Badger, L., D. Sterne, D. Sherman, K. Walker, et S. Haghighat (1995). Practical domain and type enforcement for UNIX. In *Proceedings of the IEEE Symposium on Research in Security and Privacy*, Oakland, CA, pp. 66–77. IEEE Computer Society, Technical Committee on Security and Privacy : IEEE Computer Society Press.

Bell, D. et L. LaPadula (1973). Secure Computer Systems : a Mathematical Model. Technical Report MTR-2547 (Vol. II), MITRE Corp., Bedford, MA.

Brewer, D. F. C. et M. J. Nash (1989). The chinese wall security policy. In *Proc. IEEE Symposium on Security and Privacy*, pp. 206–214.

Denning, D. et P. Denning (1977). Certification of programs for secure information flow. *Commun. ACM 20*(7), 504–513.

Denning, D. E. (1976a). A lattice model of information flow. *Communications of the ACM 19*(5), 236–243.

Denning, D. E. (1976b). On the derivation of lattice structured information flow policies. Technical Report TR-179, Dep. of Computer Science, Purdue U., W. Lafayette, Ind.

Foley, S. (1990). Secure information flow using security groups. In *IEEE Computer Security Foundations Workshop CSFW*, pp. 62–72.

Harrison, M., W. Ruzzo, et J. Ullman (1976). Protection in operating systems. *Communications of the ACM 19*, 461–471.

Hiet, G., L. Mé, J. Zimmermann, C. Bidan, B. Morin, et V. Viet Triem Tong (2007). Détection fiable et pertinente de flux d'information illégaux. In *6th Conference on Security and Network Architectures (SARSSI)*.

Hiet, G., V. Viet Triem Tong, L. Mé, et B. Morin (2008). Policy-based intrusion detection in web applications by monitoring java information flows. In *3nd International Conference on Risks and Security of Internet and Systems (CRiSIS 2008)*.

LaPadula, L. et D. Bell (1996). Secure Computer Systems : A Mathematical Model. *Journal of Computer Security 4*, 239–263.

McLean, J. (1990). Security models and information flow. In *Proceedings of the IEEE Symposium on Research in Security and Privacy*, pp. 180–187.

Osborn, S. (2002). Information flow analysis of an RBAC system. In *7th ACM Symposium on Access Control Models and Technologies SACMAT*, pp. 163–168.

Sandhu, R. S., E. J. Coyne, H. L. Feinstein, et C. E. Youman (1996). Role-based access control models. *IEEE Computer 29*(2), 38–47.

Summary

In this paper, we formally study information flows that occur during the executions of a system implementing a classical access control mechanism. More precisely, we detail how an access control policy together defines a set of illegal information flows and induces a set of possible information flows that can occur due to the elementary flows authorised by each access. We show that these two sets may coincide for some policies and we propose a mechanism dedicated to illegal information flow detection that can be useful in others cases. Finally, we describe two real implementations, at two levels of granularity, of our illegal flow detection mechanism: one for the Linux operating system and one for the Java Virtual Machine. We show that the whole approach is effective in detecting real life computer attacks.